

05-01-00

A

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventorship..... Shoff et al.
 Applicant..... WebTV Networks, Inc.
 Attorney's Docket No. MS1-430US
 Title: Compressed File System for Non-Volatile RAM

jc525 U.S. PTO
 09/560838
 04/28/00

TRANSMITTAL LETTER AND CERTIFICATE OF MAILING

To: Commissioner of Patents and Trademarks
 Washington, D.C. 20231

From: Thomas A. Jolly (509) 324-9256
 Lee & Hayes, PLLC
 421 W. Riverside Avenue, Suite 500
 Spokane, WA 99201

The following enumerated items accompany this transmittal letter and are being submitted for the matter identified in the above caption.

1. Transmittal Letter with Certificate of Mailing included.
2. PTO Return Postcard Receipt
3. Check in the Amount of \$1,294.00
4. Fee Transmittal
5. New patent application (title page plus 27 pages, including claims 1-47 & Abstract)
6. Executed Declaration
7. 5 sheets of formal drawings (Figs. 1-5)
8. Assignment w/Recordation Cover Sheet

Large Entity Status ☒ [x]

Small Entity Status ☐ []

The Commissioner is hereby authorized to charge payment of fees or credit overpayments to Deposit Account No. 12-0769 in connection with any patent application filing fees under 37 CFR 1.16, and any processing fees under 37 CFR 1.17.

Date: 4-28-2000

By: Thomas A. Jolly
 Thomas A. Jolly
 Reg. No. 39,241

CERTIFICATE OF MAILING

I hereby certify that the items listed above as enclosed are being deposited with the U.S. Postal Service as either first class mail, or Express Mail if the blank for Express Mail No. is completed below, in an envelope addressed to The Commissioner of Patents and Trademarks, Washington, D.C. 20231, on the below-indicated date. Any Express Mail No. has also been marked on the listed items.

Express Mail No. (if applicable) EL580804263

Date: 4/28/2000

By: Lori A. Vierra
 Lori A. Vierra

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

Compressed File System For Non-Volatile RAM

Inventors:

Daniel J. Shoff

Jun Liu

&

John C. Southmayd

TECHNICAL FIELD

This invention relates generally to data storage mechanisms and, more particularly, to methods and arrangements for providing a compressed file system in non-volatile random access memory (NVRAM) or other like data storage mechanism.

BACKGROUND

During normal operations, information is stored within a computing device in the form of data. For example, in a typical personal computer (PC) data can be temporarily stored in a system or primary memory consisting of random access memory (RAM). Further data may be stored in a secondary memory, such as, a hard drive, a floppy drive, optical drive, etc. In such configurations, the secondary memory provides a non-volatile data storage alternative to the primary memory since the data stored in the primary memory would usually be lost if a power failure occurred. Solid-state non-volatile RAM (NVRAM), which is capable of retaining the stored data even after a power failure, is currently available but tends to be cost prohibitive for most PCs.

There are some computing devices, however, that can benefit from the use of NVRAM. One example is a set top box (STB) that is used to provide entertainment and related information services to subscribing users. Very little data storage is typically needed to provide such services, as compared to a full-fledged PC application. Thus, all or part of the primary memory in the STB can be NVRAM. This essentially eliminates the need to provide a secondary memory, such as, a hard drive, or the like. This can reduce the overall complexity, size and cost of the STB, for example.

Since NVRAM tends to be relatively expensive when compared to RAM and/or conventional secondary memory, there is an attendant need to significantly reduce or otherwise minimize the amount of NVRAM required within a given computing device.

SUMMARY

Methods and arrangements are provided that significantly reduce or otherwise minimize the amount of NVRAM required within a given computing device.

For example, a novel data structure and management scheme are provided in a manner that allows an NVRAM sector-based memory to appear as providing significantly more storage space than it physically has. This is accomplished by mapping a higher number of virtual sectors to a fewer number of physical sectors. Data written to a plurality of virtual sectors is compressed and written to physical sector(s). The information needed to associate the virtual and physical sectors can be maintained in a virtual sector table within less expensive RAM. If power is lost and the virtual sector table is no longer available in the RAM, then on power-up the virtual sector table is recreated based in information that is imbedded within the stored data structure in physical sectors of the NVRAM. The scheme promotes data integrity by carefully controlling the compression and decompression processes and providing data and operational step backup information to insure that data within the NVRAM is not lost by a sudden power loss, etc.

1 The above stated needs are met, for example, by a method that includes
2 receiving file system data from an operating system, storing the file system data in
3 a plurality of reserved sectors within a non-volatile memory, compressing the file
4 system data stored within in the plurality of reserved sectors to create a
5 compressed data block, and storing the compressed data block in at least one
6 physical subsector within the non-volatile memory. In this manner the operating
7 system can be provided with a plurality of operatively accessible virtual sectors
8 resulting in a virtual memory capacity that exceeds the actual physical capacity of
9 the non-volatile memory.

10 The method may further include, therefore, mapping the plurality of virtual
11 sectors to at least one physical subsector through a Virtual Sector Table (VST)
12 stored in a volatile memory and presenting the operating system with the VST. A
13 Sector Allocation Table (SAT) within the volatile memory can be used to map
14 physical subsectors to the VST. The SAT can be generated based at least on a
15 unique group identifier that is stored in each physical subsector associated with
16 storing the compressed data block. For example, the SAT can be generated during
17 a device initialization time.

18 The step of storing the compressed data block in at least one physical
19 subsector within the non-volatile memory may further include associating each
20 physical subsector with a unique group identifier. The step of storing the
21 compressed data block in at least one physical subsector within the non-volatile
22 memory may further include writing each physical subsector associated with the
23 compressed data block to the non-volatile memory in an a sequential order, but not
24 necessarily a contiguous order.
25

1 In still other implementations, the step of storing the compressed data block
2 in at least one physical subsector within the non-volatile memory further includes,
3 maintaining input/output (I/O) operation status information within the non-volatile
4 memory, during on-going I/O operations, that identifies the status of on-going I/O
5 operations with respect to data stored within non-volatile memory.

6 An arrangement for use in providing an application access to a non-volatile
7 memory is also provided. The arrangement includes an operating system and a
8 device driver. Here, the operating system is configured to exchange input/output
9 (I/O) requests with the application and exchange corresponding file system
10 requests with the device driver. The device driver is configured to store the file
11 system data received from the operating system in a plurality of reserved sectors
12 within the non-volatile memory, compress the file system data stored within in the
13 plurality of reserved sectors to create a compressed data block, and store the
14 compressed data block in at least one physical subsector within the non-volatile
15 memory. This arrangement can be included along with a processor configured to
16 run the operating system and the device driver, and a non-volatile memory that is
17 operatively coupled to the processor as part of a set top box or other like
18 computing device.

19 A non-volatile computer-readable medium having stored thereon a data
20 structure, is also provided. The data structure includes a raw sector map data and a
21 plurality of raw sectors as identified in the raw sector map that are configurable to
22 store uncompressed data during input/output (I/O) operations, I/O operation status
23 data identifying on-going I/O operation, and a plurality of subsectors contiguously
24 arranged and configured to support the identified on-going I/O operations by
25 storing compressed data blocks derived from the uncompressed data.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete understanding of the various methods and arrangements of the present invention may be had by reference to the following detailed description when taken in conjunction with the accompanying drawings wherein:

Fig. 1 is a block diagram depicting an exemplary computing device having a non-volatile random access memory (NVRAM).

Fig. 2 is a block diagram depicting an exemplary functional arrangement of a computing device as in Fig. 1.

Fig. 3 is a functional block diagram depicting an arrangement for compressing file system data into an NVRAM as in Fig. 1.

Fig. 4 is an illustrative diagram depicting the layout of an NVRAM as in Fig. 1.

Fig. 5 is an illustrative/functional block diagram file system data being compressed and stored in an NVRAM as in Fig. 1.

DETAILED DESCRIPTION

The following description describes various methods and arrangements that are implemented in a computing device running Microsoft® Windows® CE. This is by way of example only. Those skilled in the art will clearly recognize that the various methods and arrangements as described below and shown in the Drawings can be used in any type of computing device having any type of operating system that is configured to read/write data from/to a non-volatile memory arrangement.

Windows® CE is a scalable operating system (herein after, referred to as the “ OS”) that is designed for a variety of embedded systems and products. Its multithreaded, multitasking, fully preemptive operating environment is designed specifically for hardware with limited resources, such as consumer electronic devices, specialized industrial controllers, and embedded communications devices. The OS also supports various hardware peripherals, devices, and networking systems. These include keyboards, mouse devices, touch panels, serial ports, Ethernet connections, modems, universal serial bus (USB) devices, audio devices, parallel ports, printer devices, and storage devices, such as PC Cards.

With this in mind, Fig. 1 is a functional block diagram depicting an exemplary computing device 20 having at least one processor 22 that is operatively coupled to a memory 24 through at least one bus 26. Processor 22 is further operatively coupled to at least one input/output device 28 through bus 26. In this example memory 24 includes read only memory (ROM) 30, non-volatile random access memory (NVRAM) 32, and random access memory (RAM) 34. During normal operations, a sector allocation table (SAT) 36, a virtual sector table (VST) 38 and a Group ID table 40 are generated and stored in RAM 34. These tables and the information in them are described in greater detail in subsequent sections. Those skilled in the art will recognize that computing device 20 may be implemented through the use of one or more integrated circuits, etc.

Fig. 2 is a functional block diagram depicting an exemplary functional arrangement 60 of computing device 20, for example, as in Fig. 1. Here, an application program 62 is operatively configured to interface with an operating system (OS) 64 and support reading and writing operations to a memory. OS 64 is

1 in turn operatively configured to interface with a device driver, in this case, a
2 block device driver, that further supports the read/write operation.

3 Block device drivers are provided for data storage devices that allow data to
4 be read or written only in blocks of a fixed size. Block devices do not allow
5 individual bytes of data to be read or written. Block sizes tend to be one or a few
6 kilobytes; some common sizes are 512 bytes, 1 KB, 2 KB, and 4 KB. Block
7 devices are ideally suited to mass storage and persistent storage applications, such
8 as disk and tape drives, or NVRAM.

9 As shown in Fig. 2, OS 64 includes a FAT File System (FATFS) 66 to
10 support an “NVFile” device driver 68. FATFS 66 essentially implements a logical
11 file system and provides an abstraction between files in the application name space
12 and devices in the device name space. For example, FATFS can exchange “Read
13 File ()” 70, “Write File ()” 72 and “IOCTL_DISK_FORMAT_VOLUME 74, as
14 applicable, with application 62. The non-volatile file (“NVFile”) device driver 68,
15 in this example a block device driver, is responsible for guaranteeing safe I/O
16 operations even when device 20 is interrupted by a power cycle. NVFile 68
17 exchanges a read request “DISK_IOCTL_READ” 76 and write request
18 “DISK_IOCTL_WRITE” 78, as applicable, with FATFS 66. In the above
19 example, both FATFS 66 and NVFile 68 are implemented as data link lists (e.g.,
20 dll modules).

21 NVFile device driver 68 creates a storage device of NVRAM 32 to persist
22 raw data from the FATFS file system. As described herein, it is possible and
23 advantageous to extend the available storage space by compressing file system
24 data into NVRAM 32. This can be accomplished without making substantial
25 changes to the FATFS layer supplied by the core OS 64.

1 The successful implementation of a compressed file system as described
2 herein takes advantage of conventional software compression schemes, such as the
3 LZX library from the OS. This exemplary software compression scheme tends to
4 yields an average compression ratio of approximately 4:1.

5 The NVFile storage device in this example can be addressed by FAT-12
6 implementation using 512-byte sectors, and one sector per cluster, for example.
7 Here, FATFS 66 notifies its underlying block device driver NVFile 68 when
8 formatting or file (cluster) delete operations occur. FATFS 66 also performs a
9 low-level format whenever a media format operation is performed.
10 Communication between FATFS 66 and NVFile 68 insures correct reporting of
11 “disk full” status and the existence of free space. For example, FATFS 66 notifies
12 NVFile 68 when clusters are deleted, to insure that free space is reported
13 accurately and free clusters can be properly identified as free space.

14 Reference is now made to Fig. 3, which is a functional block diagram
15 depicting an arrangement for compressing file system data into NVRAM 32 as in
16 Fig. 1. As depicted, FATFS 66 provides a write request (78) that specifies a buffer
17 “pData” (e.g., a block of data) and a “Sector XXX”. Sector XXX is then mapped
18 to a “virtual sector” within VST 38. Since the actual physical storage capacity of
19 NVRAM 32 is less than that supported by VST 38, a compression stage 80 is
20 introduced in NVFile to compress the “pData” and store a corresponding
21 compressed block of data on NVRAM 32.

22 This process saves NVRAM for data and is designed to avoid wasting
23 valuable NVRAM by persisting too much organizational information or tables of
24 pointers. Thus, in this example, tables and pointers are almost exclusively
25 maintained in RAM 34 rather than in NVRAM 32. Nevertheless, atomic/safe

1 operations are provided to maintain data and organizational integrity across
2 power-cycles. As will be appreciated, the methods and arrangements presented
3 herein also tend to avoid heap fragmentation.

4 Fig. 4 is an illustrative diagram depicting an exemplary layout of NVRAM
5 32. Here, NVRAM 32 includes a raw sector map 100, an I/O operation status 102,
6 a plurality of reserved raw sectors 104 (numbered 0-6), and a plurality of
7 subsectors 106 (numbered 0-N).

8 Let us assume, in this example, that NVRAM 32 is a 100-k byte memory
9 within a set top box. Here, the beginning of the 100-k block of NVRAM 32 starts
10 with a 28-byte raw sector map 100 (one DWORD for each reserved sector), then a
11 series of seven 512-byte raw uncompressed reserved sectors 104. Following the
12 end of the reserved sectors 104 are the 128-byte subsectors 106 to store
13 compressed blocks of data from the file system.

14 In raw sector map 100, either a virtual sector number or an “unused” value
15 is used to indicate what is stored in each reserved sector 104.

16 An I/O operation status 102 is provided between raw sector map 100 and
17 the first reserved sector 104. I/O operation status 102 stores the current state of
18 NVFile 68 in case of a power failure or normal power cycle during an I/O
19 operation. For example, I/O operation status 102 can include a flag indicating the
20 type of operation (write, delete, compress, etc.) and then some additional data
21 depending on the context/stage of the operation.

22 Since, as described below, compression will normally occur after every
23 fourth sector-write in this exemplary arrangement, seven reserved sectors 104 are
24 employed to cache data in the very likely event that a power cycle will occur while
25

1 the data are awaiting compression. Thus, all data from the file system is persisted
2 somewhere while it is being moved.

3 The seven reserved sectors 104 include additional space for use during
4 delete operations. For example, as part of a delete operation the compressed block
5 containing the deleted cluster must be decompressed and reassembled with a new
6 fourth sector. To provide for safe/atomic operation, all four sectors of that
7 particular compressed block are stored in reserved sectors 104 until such time as
8 the delete operation and recompression has been successfully completed.

9 The remaining space in NVRAM 32 is divided into 128-byte subsectors
10 where compressed data blocks are stored. Each 128-byte subsector will begin
11 with a group ID 108 (or marked "unused"), as shown in greater detail in
12 subsectors 106a and 106b. Group ID 108 will identify which subsectors 106
13 belong to which compressed block, since they will not necessarily be written
14 contiguously. In addition, as depicted in subsector 106a, the first subsector of a
15 compressed block within NVRAM 32 will have, following the group ID 108, four
16 virtual sector numbers 110 to identify where the data belongs in the real file
17 system.

18 Fig. 5 is an illustrative/functional block diagram file system data being
19 compressed and stored in subsectors 106. Here, it is assumed that write requests
20 #1, #2, and #3 have each been received and stored in raw sectors 0, 1, and 2,
21 respectively. When write request #4 is subsequently received, all four data blocks
22 are provided to a data compressor function 120 to produce a compressed block of
23 data. The compressed block of data is then provided to a splitter function that
24 stores pieces A, B and C of the compressed data in unused subsectors 106.
25

1 The Sector Allocation Table (SAT) 36 is essentially the “glue” relating the
2 compressed block subsectors 106 of NVRAM 32 to the Virtual Sector Table
3 (VST) 38 that is presented to the file system. Group IDs 108 are stored in
4 compressed subsectors to allow for generation of SAT 36 at boot-up/init time.
5 SAT 36 is basically a fixed-size mirror of the compressed-block subsectors 106 in
6 NVRAM 32. There is one SAT entry for each compressed subsector 106, and
7 each one will either be marked “unused” or indicate the next compressed subsector
8 106 in the chain. The last member of a chain will point to itself to mark the end.

9 SAT information is not be stored directly to NVRAM 32, but rather
10 inferred at boot time using Group ID 108 and the Virtual Sector Numbers 110 that
11 are stored in every physical compressed subsector 106. The Group ID number and
12 the sequential order in which subsectors having a common Group ID 108 are
13 written/read, will be used to generate the compressed subsector chain information
14 in SAT 36.

15 At boot-time, for example, each NVRAM subsector 106 is read, in order
16 from the beginning to the end of the subsector storage space. The first byte of
17 each subsector contains either an “unused” value, or a Group ID 108. For each
18 unique Group ID 108, every following subsector 106 with a matching Group ID
19 108 will be associated with that first subsector 106. The resulting chain of
20 associated subsectors 106, in the order found, will eventually comprise a
21 compressed block that can be reassembled and decompressed. For the first unique
22 Group ID found, an entry is made in the SAT that is later filled in with the offset
23 address of the next subsector 106 in that group. When the final subsector 106 is
24 found (determined only by reaching the end of the NVRAM and finding no more
25

subsectors in that group), the final SAT 36 entry is marked with its own address, indicating the end of the chain.

As described above, instead of mapping FATFS sectors directly to like-sized blocks of NVRAM 32, NVFile 68 presents VST 38 to FATFS while actually storing compressed blocks of data in the actual NVRAM subsectors 106. In order to minimize NVRAM overhead of this scheme, VST 38 is generated on the fly from this information at boot time, rather than stored in NVRAM in its entirety. In addition, a small portion of uncompressed data is stored in reserved raw sectors 104 of NVRAM 32 to persist sectors that have been written but not had a chance to compress yet.

To properly generate VST 38 at boot time, it is necessary to maintain and persist (during write operations) the relationship between the compressed data blocks and the virtual sectors. This relation is stored in the table of Group IDs 40, which map virtual sectors to groups of physical compressed subsectors 106.

When a new group of four sectors is compressed, they are assigned an unused Group ID 108. A table is stored at the beginning of the compressed data block, listing the virtual sector numbers represented by the compressed block. This relationship between the Group ID, physical subsectors, and virtual sectors numbers, is maintained at runtime. During initialization, Group ID table 40 is therefore dynamically generated, and maintained during all I/O operations.

By way of example, each entry in VST 38 can be a pointer to the first SAT entry corresponding to that virtual sector. For each compressed block containing four virtual sectors of data, there will be four VSTs pointing to its first SAT entry. The I/O operation can then start from that SAT entry and simply “walk” the pointers to retrieve and reassemble the entire compressed block.

1 The purpose of Group ID 108 is to store as little organizational (i.e., non-
2 data) information as possible in NVRAM. Every virtual sector in the VST must
3 point to a compressed block, but since each compressed block contains four virtual
4 sectors, otherwise unrelated virtual sectors will share the Group ID they reside in.

5 When four sectors are compressed, the new compressed block is assigned
6 the next-unused Group ID 108 from the Group ID table 40 in RAM 34. Group ID
7 108 itself is saved as the first byte in each physical sector that the compressed
8 block gets stored in.

9 During boot-up time, Group ID table 40 is re-created based on the stored
10 Group ID information. This can be accomplished by looping iteratively through
11 the NVRAM physical subsectors, matching up all of the sectors with others
12 sharing the same Group ID 108 to reestablish the associative relationship. Then,
13 the virtual sector numbers and Group IDs can be linked in the table in RAM 34.

14 Fig. 3 demonstrates how a sector write request from the file system is
15 routed through VST 38, and the data buffer is compressed and stored in a
16 compressed block of NVRAM 32.

17 As shown in the example of Fig. 5, to increase the efficiency of the data
18 compression, four sectors are compressed at a time rather than just one. However,
19 there is no guarantee that write operations will come in four-sector increments. As
20 a result, a buffer is persisted which can store uncompressed data that is waiting to
21 be compressed. The first "line" of physical NVRAM (raw sectors "0", "1", and
22 "2") is reserved for three raw sectors, and once those are filled they can be
23 combined with the fourth sector of incoming data, to produce a compressed block.

24 The compressed block will then be split so that it can fit into a number of
25 smaller physical memory subsectors 106. How many subsectors are required will

1 depend on the size of the final compressed block. Pieces of the compressed block
2 are then copied into NVRAM 32 and stored persistently.

3 The location of free physical memory sectors may be determined using an
4 iterative loop to find the next available unused subsector, if one is available. Since
5 it is difficult to know ahead of time the final size of a compressed block, it is
6 preferable to allow for fragmentation to occur so as to make better use of the
7 available memory space.

8 As part of the write operation, NVFile is also responsible for maintaining
9 the integrity of the Group ID table 40. As well as updating the dynamic Group ID
10 table 40 in RAM 34 to reflect the new write operation, it must also store the
11 correct Group ID 108 info in the subsectors 106 that are written.

12 After the compressed block is split into subsectors 106, they must be kept
13 in order so that the original compressed block can be recreated later for
14 decompression. From the beginning to the end of NVRAM 32, subsectors 106
15 belonging to any compressed block will be in order; there will be no actual
16 pointers stored in NVRAM 32 for purposes of maintaining subsector chains.

17 An incoming Read I/O operation will result in a hit on VST 38, where
18 actual file system sectors correspond to locations of NVRAM compressed blocks.
19 Since multiple virtual sectors are stored in a single compressed block, and the
20 compressed block must be decompressed to retrieve data, the entire compressed
21 block must be reassembled. Since this is true for any virtual sector residing within
22 the block, virtual sectors will all point to the beginning of their respective
23 compressed blocks. Further addressing is stored in the compressed block header
24 and decoded after decompression.

1 When a virtual sector hit point to any physical memory portion within a
2 compressed block, the first memory portion is read, and the chain followed to each
3 memory portion comprising the compressed block. The compressed block can
4 then be reassembled in a RAM memory buffer, decompressed, and the resulting
5 raw data returned to the file system.

6 There is a chance that a VST hit will point to an uncompressed raw sector
7 104 in the reserved line of physical NVRAM 32. In these cases, the uncompressed
8 data will simply be read straight through and returned to the file system.

9 When a virtual sector is deleted by FATFS, the compressed block in which
10 it resides is decompressed, and reassembled without the deleted data. If one or
11 more uncompressed sectors are already waiting in the reserved area 104, one of
12 them can be combined with the remaining three from the compressed block to
13 write a new block of four sectors. However, the existing compressed block cannot
14 be deleted until a new one is written. In order to insure safe/atomic operation, it is
15 important to reserve four additional uncompressed sectors 104 (i.e., raw sectors
16 "3", "4", "5", and "6") in the worst case but likely event that the first line of raw
17 sectors 104 (i.e., raw sectors "0", "1", and "2") are already full when a cluster is
18 deleted.

19 Deleted virtual sectors will be marked as unused in VST 38 and SAT 36,
20 and their former Group ID 108 is marked "unused".

21 A common scenario is to have a FATFS sector re-written with new data.
22 This operation takes place in much the same way as the aforementioned delete
23 operation. The main difference is that instead of finding a new virtual sector to
24 replace the deleted one, a new compressed block and subsector chain are stored
25 using the new data from the updated sector.

1 FATFS 66 may also be modified to notify NVFile 68 via an IOCTL when a
2 format operation takes place. This way NVFile can mark all of the clusters as
3 unused and reset all of the tables. It is important that this notification take place so
4 that the status of the used versus free sectors is consistent between the file system
5 and NVFile 68.

6 It should be understood that the methods and arrangements described herein
7 are not limited to data compression stages having compression ratios of 4:1,
8 memory layouts having seven reserved sectors, raw sectors of 512 bytes, or
9 compressed sectors of 128 bytes. For example, one or more of these parameters
10 can be higher or lower depending upon the computing device.

11 Thus, although some preferred embodiments of the various methods and
12 arrangements of the present invention have been illustrated in the accompanying
13 Drawings and described in the foregoing Detailed Description, it will be
14 understood that the invention is not limited to the exemplary embodiments
15 disclosed, but is capable of numerous rearrangements, modifications and
16 substitutions without departing from the spirit of the invention as set forth and
17 defined by the following claims.

1 **CLAIMS**

2
3 What is claimed is:

4
5 1. A method comprising:
6 receiving file system data;
7 storing the file system data in a plurality of reserved sectors within a non-
8 volatile memory;
9 compressing the file system data stored within in the plurality of reserved
10 sectors to create a compressed data block; and
11 storing the compressed data block in at least one physical subsector within
12 the non-volatile memory.

13
14 2. The method as recited in Claim 1, wherein receiving file system data
15 further includes presenting an operating system with a plurality of operatively
16 accessible virtual sectors resulting in a virtual memory capacity that exceeds the
17 actual physical capacity of the non-volatile memory.

18
19 3. The method as recited in Claim 2, wherein storing the compressed data
20 block at least one physical subsector within the non-volatile memory further
21 includes mapping the plurality of virtual sectors to at least one physical subsector
22 through a Virtual Sector Table (VST) stored in a volatile memory and presenting
23 the operating system with the VST.

1 4. The method as recited in Claim 3, wherein mapping the plurality of virtual
2 sectors to at least one physical subsector through the Virtual Sector Table (VST)
3 further includes providing a Sector Allocation Table (SAT) within the volatile
4 memory, the SAT mapping the physical subsectors to the VST.

5
6 5. The method as recited in Claim 4, wherein providing a Sector Allocation
7 Table (SAT) within the volatile memory further includes generating the SAT based
8 at least on a unique group identifier that is stored in each physical subsector
9 associated with storing the compressed data block.

10
11 6. The method as recited in Claim 5, wherein the Sector Allocation Table
12 (SAT) is generated during a device initialization time.

13
14 7. The method as recited in Claim 1, wherein storing the compressed data
15 block in at least one physical subsector within the non-volatile memory further
16 includes associating each physical subsector with a unique group identifier.

17
18 8. The method as recited in Claim 7, wherein storing the compressed data
19 block in at least one physical subsector within the non-volatile memory further
20 includes writing each physical subsector associated with the compressed data
21 block to the non-volatile memory in an a sequential order, but not necessarily a
22 contiguous order.

1 9. The method as recited in Claim 1, wherein storing the compressed data
2 block in at least one physical subsector within the non-volatile memory further
3 includes associating a first physical subsector with at least one virtual sector
4 identifier.

5
6 10. The method as recited in Claim 1, wherein storing the compressed data
7 block in at least one physical subsector within the non-volatile memory further
8 includes, maintaining input/output (I/O) operation status information within the
9 non-volatile memory during on-going I/O operations.

10
11 11. The method as recited in Claim 1, wherein the non-volatile memory further
12 includes a raw sector map configured to identify a plurality of reserved sectors
13 within the non-volatile memory.

14
15 12. The method as recited in Claim 11, wherein the non-volatile memory
16 further includes an input/output (I/O) operation status area that identifies the status
17 of on-going I/O operations with respect to data stored within non-volatile memory.

18
19 13. The method as recited in Claim 11, wherein the non-volatile memory
20 further includes a plurality of contiguously arranged reserved sectors.

21
22 14. The method as recited in Claim 11, wherein the non-volatile memory
23 further includes a plurality of contiguously arranged physical subsectors.

1 15. The method as recited in Claim 14, wherein each of the physical subsectors
2 includes a first portion that includes a group identifier.

3
4 16. The method as recited in Claim 14, wherein the group identifier identifies
5 that the physical subsector is unused.

6
7 17. A computer-readable medium having computer-executable instructions for
8 performing steps comprising:

9 receiving file system data;

10 storing the file system data in a plurality of reserved sectors within a non-
11 volatile memory;

12 compressing the file system data stored within in the plurality of reserved
13 sectors to create a compressed data block; and

14 storing the compressed data block in at least one physical subsector within
15 the non-volatile memory.

16
17 18. The computer-readable medium as recited in Claim 17, wherein receiving
18 file system data further includes presenting an operating system with a plurality of
19 operatively accessible virtual sectors resulting in a virtual memory capacity that
20 exceeds the actual physical capacity of the non-volatile memory.

1 19. The computer-readable medium as recited in Claim 18, wherein storing the
2 compressed data block at least one physical subsector within the non-volatile
3 memory further includes mapping the plurality of virtual sectors to at least one
4 physical subsector through a Virtual Sector Table (VST) stored in a volatile
5 memory and presenting the operating system with the VST.

6
7 20. The computer-readable medium as recited in Claim 19, wherein mapping
8 the plurality of virtual sectors to at least one physical subsector through the Virtual
9 Sector Table (VST) further includes providing a Sector Allocation Table (SAT)
10 within the volatile memory, the SAT mapping the physical subsectors to the VST.

11
12 21. The computer-readable medium as recited in Claim 20, wherein providing a
13 Sector Allocation Table (SAT) within the volatile memory further includes
14 generating the SAT based at least on a unique group identifier that is stored in each
15 physical subsector associated with storing the compressed data block.

16
17 22. The computer-readable medium as recited in Claim 21, wherein the Sector
18 Allocation Table (SAT) is generated during a device initialization time.

19
20 23. The computer-readable medium as recited in Claim 17, wherein storing the
21 compressed data block in at least one physical subsector within the non-volatile
22 memory further includes associating each physical subsector with a unique group
23 identifier.

1 24. The computer-readable medium as recited in Claim 23, wherein storing the
2 compressed data block in at least one physical subsector within the non-volatile
3 memory further includes writing each physical subsector associated with the
4 compressed data block to the non-volatile memory in an a sequential order, but not
5 necessarily a contiguous order.

6
7 25. The computer-readable medium as recited in Claim 17, wherein storing the
8 compressed data block in at least one physical subsector within the non-volatile
9 memory further includes associating a first physical subsector with at least one
10 virtual sector identifier.

11
12 26. The computer-readable medium as recited in Claim 17, wherein storing the
13 compressed data block in at least one physical subsector within the non-volatile
14 memory further includes, maintaining input/output (I/O) operation status
15 information within the non-volatile memory during on-going I/O operations.

16
17 27. The computer-readable medium as recited in Claim 17, wherein the non-
18 volatile memory further includes a raw sector map configured to identify a
19 plurality of reserved sectors within the non-volatile memory.

20
21 28. The computer-readable medium as recited in Claim 27, wherein the non-
22 volatile memory further includes an input/output (I/O) operation status area that
23 identifies the status of on-going I/O operations with respect to data stored within
24 non-volatile memory.
25

1 29. The computer-readable medium as recited in Claim 27, wherein the non-
2 volatile memory further includes a plurality of contiguously arranged reserved
3 sectors.

4
5 30. The computer-readable medium as recited in Claim 27, wherein the non-
6 volatile memory further includes a plurality of contiguously arranged physical
7 subsectors.

8
9 31. The computer-readable medium as recited in Claim 30, wherein each of the
10 physical subsectors includes a first portion that includes a group identifier.

11
12 32. The computer-readable medium as recited in Claim 30, wherein the group
13 identifier identifies that the physical subsector is unused.

14
15 33. An arrangement for use in providing an application access a non-volatile
16 memory, the arrangement comprising:

17 an operating system; and

18 a device driver, wherein the operating system is configured to exchange
19 input/output (I/O) requests with the application and exchange corresponding file
20 system requests with the device driver, and wherein the device driver is configured
21 to store the file system data received from the operating system in a plurality of
22 reserved sectors within the non-volatile memory, compress the file system data
23 stored within in the plurality of reserved sectors to create a compressed data block,
24 and store the compressed data block in at least one physical subsector within the
25 non-volatile memory.

1
2 34. The arrangement as recited in Claim 33, wherein the device driver is further
3 configured to present the operating system with a plurality of operatively
4 accessible virtual sectors resulting in a virtual memory capacity that exceeds the
5 actual physical capacity of the non-volatile memory.
6

7 35. The arrangement as recited in Claim 34, wherein the device driver is further
8 configured to map the plurality of virtual sectors to at least one physical subsector
9 through a Virtual Sector Table (VST) stored in a volatile memory and present the
10 operating system with the VST.
11

12 36. The arrangement as recited in Claim 35, wherein the device driver is further
13 configured to store a Sector Allocation Table (SAT) within the volatile memory,
14 the SAT mapping the physical subsectors to the VST.
15

16 37. The arrangement as recited in Claim 36, wherein the device driver is further
17 configured to generate the SAT based at least on a unique group identifier that is
18 stored in each physical subsector associated with storing the compressed data
19 block.
20

21 38. The arrangement as recited in Claim 37, wherein the device driver is further
22 configured to generate the Sector Allocation Table (SAT) during a device
23 initialization time.
24
25

1 39. The arrangement as recited in Claim 33, wherein the device driver is further
2 configured to associate each physical subsector with a unique group identifier.

3
4 40. The arrangement as recited in Claim 39, wherein the device driver is further
5 configured to write each physical subsector associated with the compressed data
6 block to the non-volatile memory in an a sequential order, but not necessarily a
7 contiguous order.

8
9 41. The arrangement as recited in Claim 33, wherein the device driver is further
10 configured to associate a first physical subsector with at least one virtual sector
11 identifier.

12
13 42. The arrangement as recited in Claim 33, wherein the device driver is further
14 configured to maintain input/output (I/O) operation status information within the
15 non-volatile memory during on-going I/O operations.

16
17 43. The arrangement as recited in Claim 33, further comprising:
18 a processor configured to run the operating system and the device driver;
19 and
20 a non-volatile memory operatively coupled to the processor.

21
22 44. The arrangement as recited in Claim 43, wherein the processor and non-
23 volatile memory are part of a set top box.
24
25

1 45. A non-volatile computer-readable medium having stored thereon a data
2 structure, comprising:

3 raw sector map data;

4 a plurality of raw sectors as identified in the raw sector map that are
5 configurable to store uncompressed data during input/output (I/O) operations;

6 I/O operation status data identifying on-going I/O operations; and

7 a plurality of subsectors contiguously arranged and configured to support
8 the identified on-going I/O operations by storing compressed data blocks derived
9 from the uncompressed data.

10
11 47. The non-volatile computer-readable medium of Claim 46, wherein each of
12 the plurality of subsectors further includes a group identifier that is uniquely
13 associated with the associated compressed data block.

1 **ABSTRACT**

2 Methods and arrangements are provided that significantly reduce or
3 otherwise minimize the amount of NVRAM required within a given computing
4 device. For example, a novel data structure and management scheme are provided
5 in a manner that allows an NVRAM sector-based memory to appear as providing
6 significantly more storage space than it physically has. This is accomplished by
7 mapping a higher number of virtual sectors to a fewer number of physical sectors.
8 Data written to a plurality of virtual sectors is compressed and written to physical
9 sector(s). The information needed to associate the virtual and physical sectors can
10 be maintained in a virtual sector table within less expensive RAM. If power is lost
11 and the virtual sector table is no longer available in the RAM, then on power-up
12 the virtual sector table is recreated based in information that is imbedded within
13 the stored data structure in physical sectors of the NVRAM. The scheme
14 promotes data integrity by carefully controlling the compression and
15 decompression processes and providing data and operational step backup
16 information to insure that data within the NVRAM is not lost by a sudden power
17 loss, etc.

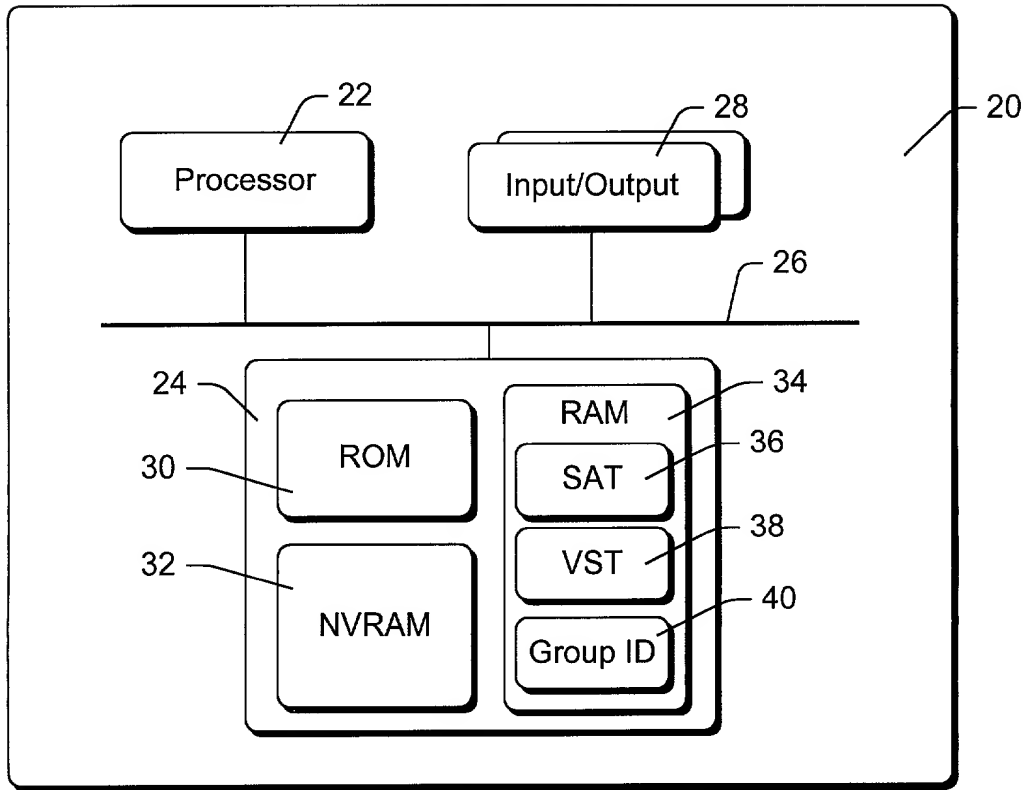
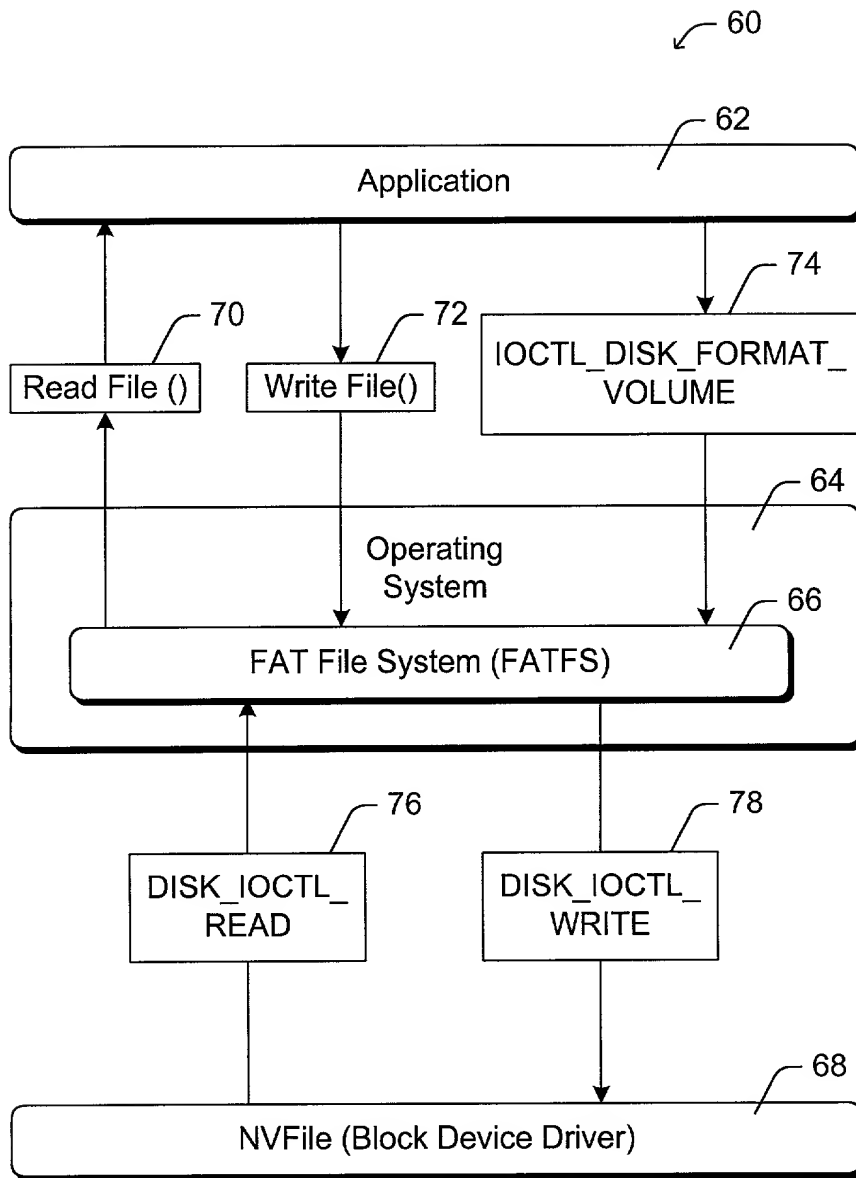
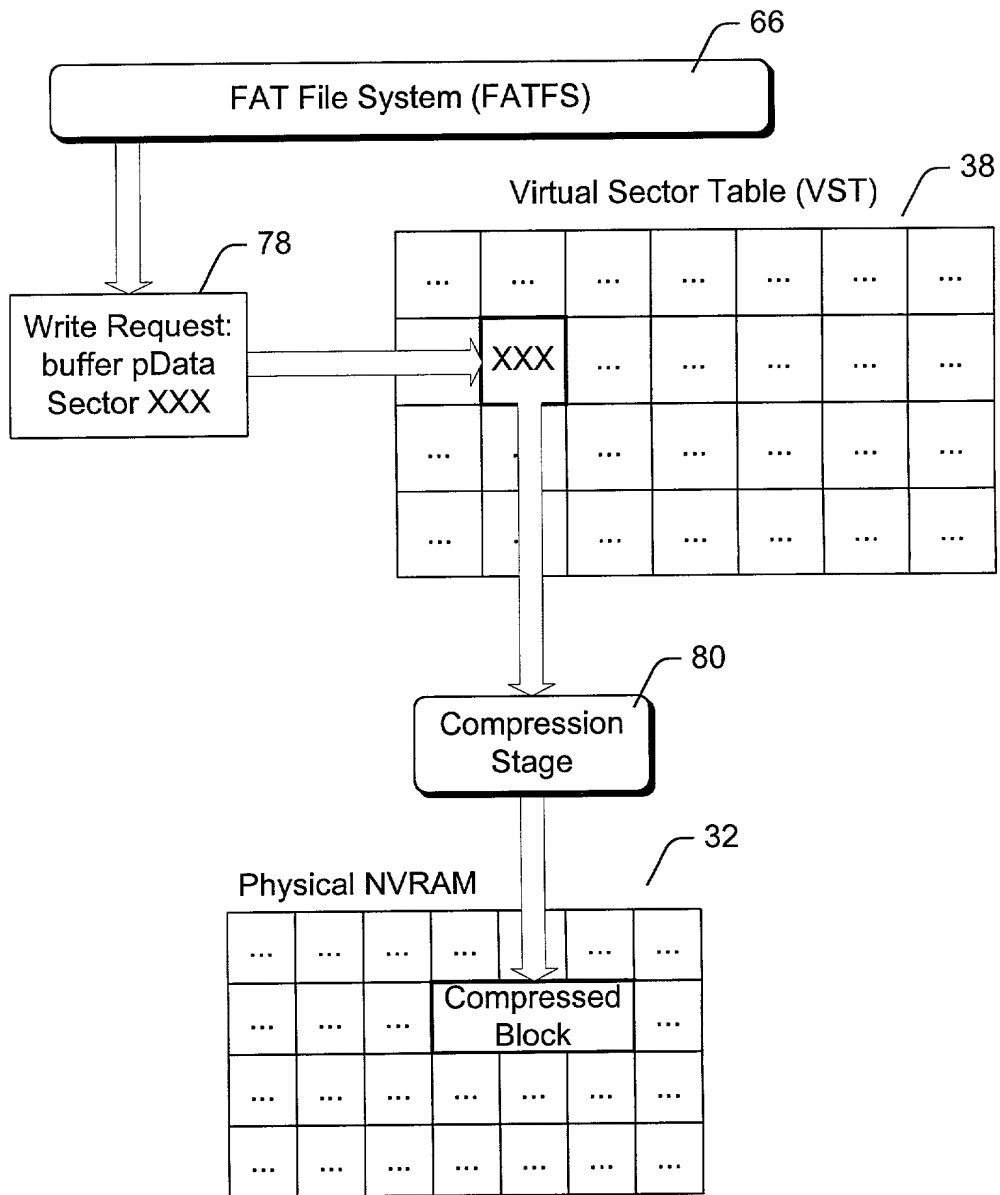
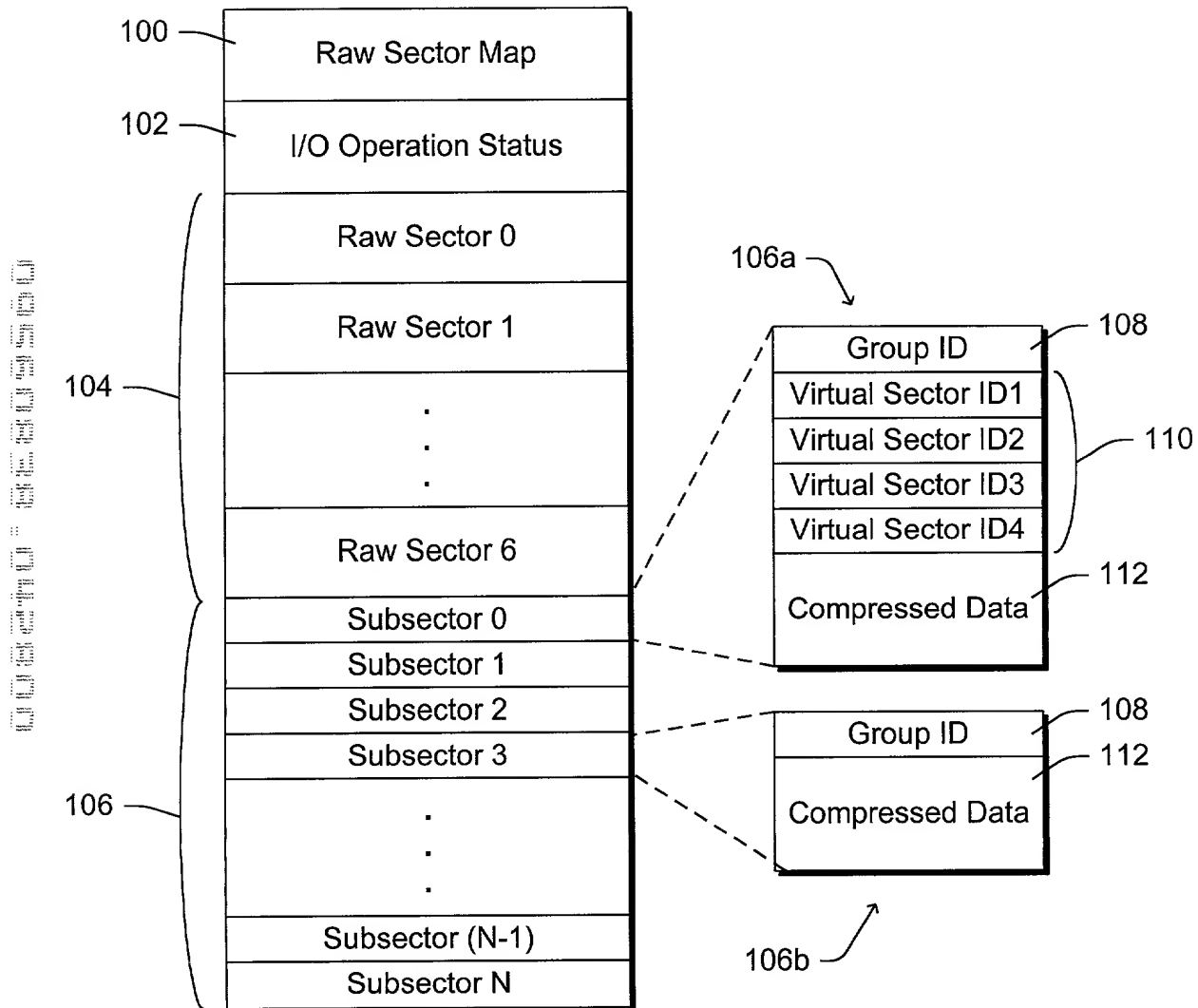


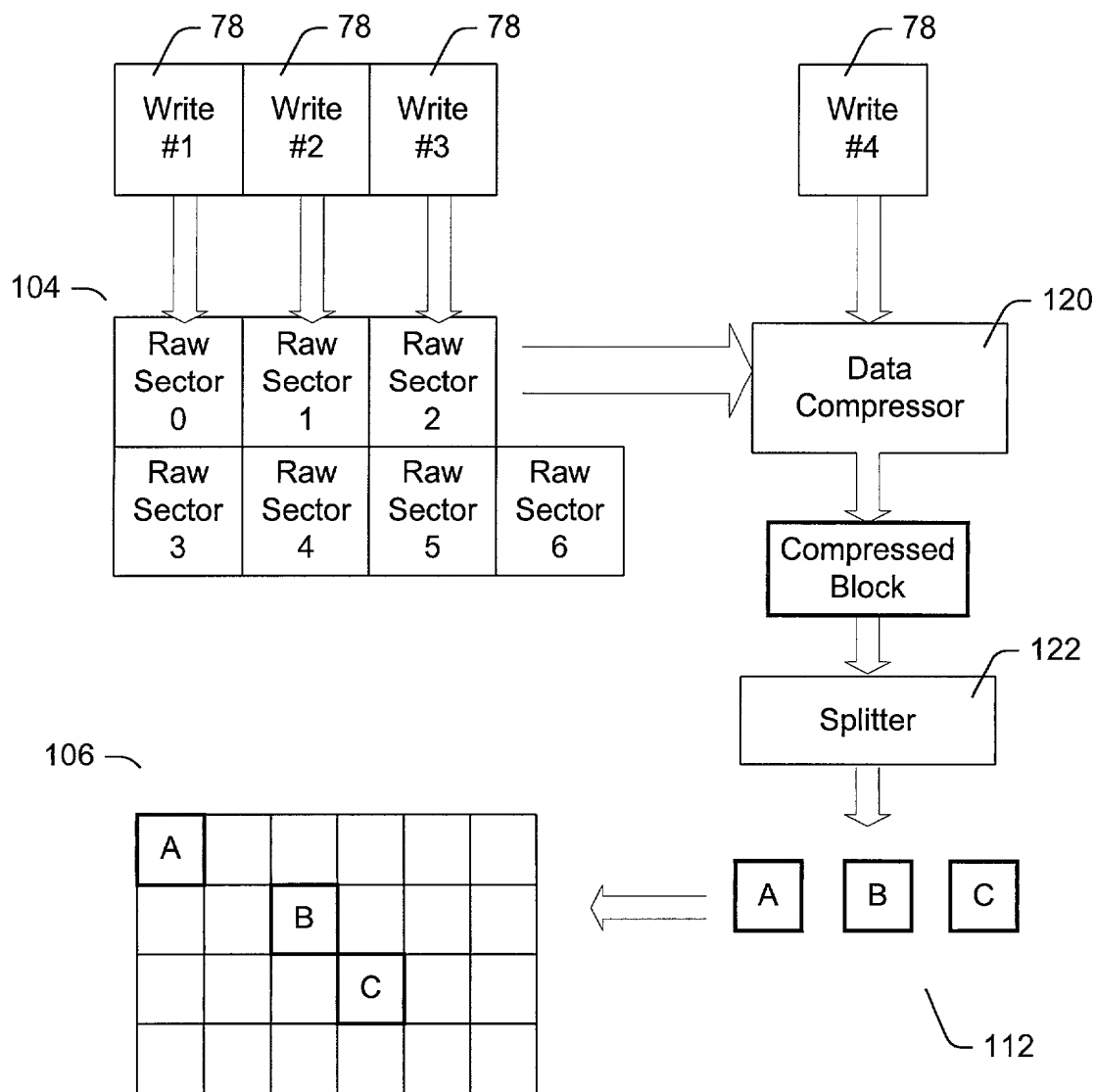
Fig. 1

*Fig. 2*

*Fig. 3*

32

*Fig. 4*

*Fig. 5*

1 **IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

2 Inventorship..... Shoff et al.
3 Applicant WebTV Networks, Inc.
4 Attorney's Docket No. MS1-430US
5 Title: Compressed File System for Non-Volatile RAM

6 **DECLARATION FOR PATENT APPLICATION**

7 As a below named inventor, I hereby declare that:

8 My residence, post office address and citizenship are as stated below next to
9 my name.

10 I believe I am the original, first and sole inventor (if only one name is listed
11 below) or an original, first and joint inventor (if plural names are listed below) of the
12 subject matter which is claimed and for which a patent is sought on the invention
13 entitled "Compressed File System for Non-Volatile RAM," the specification of
14 which is attached hereto.

15 I have reviewed and understand the content of the above-identified
16 specification, including the claims.

17 I acknowledge the duty to disclose information which is material to the
18 examination of this application in accordance with Title 37, Code of Federal
19 Regulations, § 1.56(a).

20 PRIOR FOREIGN APPLICATIONS: no applications for foreign patents or
21 inventor's certificates have been filed prior to the date of execution of this
22 declaration.

23 **Power of Attorney**

24 I appoint the following attorneys to prosecute this application and transact all
25 future business in the Patent and Trademark Office connected with this application:
26 Lewis C. Lee, Reg. No. 34,656; Daniel L. Hayes, Reg. No. 34,618; Allan T.

1 Sponseller, Reg. 38,318; Steven R. Sponseller, Reg. No. 39,384; James R.
2 Banowsky, Reg. No. 37,773; Lance R. Sadler, Reg. No. 38,605; Michael A. Proksch,
3 Reg. No. 43,021; Thomas A. Jolly, Reg. No. 39,241; David A. Morasch, Reg. No.
4 42,905; Kasey C. Christie, Reg. No. 40,559; Katie E. Sako, Reg. No. 32,628 and
5 Daniel D. Crouse, Reg. No. 32,022.

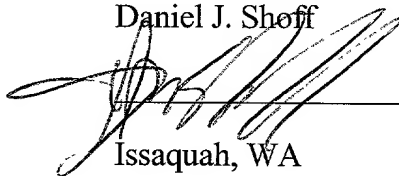
6 Send correspondence to: LEE & HAYES, PLLC, 421 W. Riverside Avenue,
7 Suite 500, Spokane, Washington, 99201. Direct telephone calls to: Thomas A. Jolly
8 (509) 324-9256.

9
10 All statements made herein of my own knowledge are true and that all
11 statements made on information and belief are believed to be true; and further that
12 these statements were made with the knowledge that willful false statements and the
13 like so made are punishable by fine or imprisonment, or both, under Section 1001 of
14 Title 18 of the United States Code and that such willful false statement may
15 jeopardize the validity of the application or any patent issued therefrom.

16
17 * * * * *

18 Full name of inventor: Daniel J. Shoff

19 Inventor's Signature



Date: 7/26/2000

20 Residence:

Issaquah, WA

21 Citizenship:

USA


22 Post Office Address:

381 Crystal Creek Cir
Issaquah, WA 98027

Full name of inventor:

Jun Liu

Inventor's Signature



Date:

4/26/2000

Residence:

Redmond, WA

Citizenship:

China

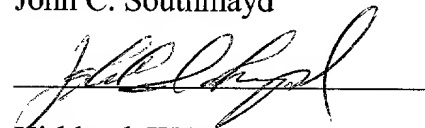
Post Office Address:

7001 Old Redmond Rd., Apt. G227
Redmond, WA 98052

Full name of inventor:

John C. Southmayd

Inventor's Signature



Date:

4/27/2000

Residence:

Kirkland, WA

Citizenship:

USA

Post Office Address:

12616 NE 144th St., Apt. K-105
Kirkland, WA 98034